

Análisis de Algoritmos y Estructuras de Datos

Práctica 7: Uso del TAD Lista

Versión 2.1

Pasos a seguir

1. Implemente el TAD *Lista* utilizando las representaciones pseudoestática y dinámica.
2. Escriba módulos que contengan las implementaciones de los subprogramas demandados en cada problema.
3. Para cada uno de los problemas, escriba un programa de prueba donde se realicen las llamadas a los subprogramas, comprobando el resultado de salida.

Ejercicios

1. Implemente el TAD *Lista Circular* partiendo de la siguiente especificación.

Definición:

Una *lista circular* es una secuencia de elementos de un mismo tipo en la que todos tienen un predecesor y un sucesor, es decir, es una secuencia sin extremos. Su *longitud* coincide con el número de elementos que la forman; si es 0, entonces la lista está vacía. Una lista circular de longitud n se puede representar de la forma

$$L = (a_1, a_2, \dots, a_n, a_1)$$

donde repetimos a_1 después de a_n para indicar que el elemento que sigue a a_n es a_1 y el anterior a éste es a_n .

Definimos una *posición* como el lugar que ocupa un elemento en la lista. La constante *POS_NULA* denota una posición inexistente.

Operaciones:

ListaCir()

Postcondiciones: Crea y devuelve una lista circular vacía.

void insertar(const Tℰ x, posicion p)

Precondiciones: $L = ()$ y p es irrelevante o bien,

$L = (a_1, a_2, \dots, a_n, a_1)$ y $1 \leq p \leq n$

Postcondiciones: Si $L = ()$, entonces $L = (x, x)$ (lista circular con un único elemento);
en caso contrario,

$L = (a_1, \dots, a_{p-1}, x, a_p, \dots, a_n, a_1)$

void eliminar(posicion p)

Precondiciones: $L = (a_1, a_2, \dots, a_n, a_1)$

$1 \leq p \leq n$

Postcondiciones: $L = (a_1, \dots, a_{p-1}, a_{p+1}, \dots, a_n, a_1)$

const Tℰ elemento(posicion p) const

Tℰ elemento(posicion p)

Precondiciones: $L = (a_1, a_2, \dots, a_n, a_1)$

$1 \leq p \leq n$

Postcondiciones: Devuelve a_p , el elemento que ocupa la posición p .

posicion buscar(const Tℰ x) const

Postcondiciones: Devuelve la posición de una ocurrencia de x en la lista. Si x no pertenece a la lista, devuelve POS_NULA.

posicion inipos() const

Postcondiciones: Devuelve una posición indeterminada de la lista. Si la lista está vacía, devuelve POS_NULA. Esta operación se utilizará para inicializar una variable de tipo *posicion*.

posicion siguiente(posicion p) const

Precondiciones: $L = (a_1, a_2, \dots, a_n, a_1)$

$1 \leq p \leq n$

Postcondiciones: Devuelve la posición siguiente a p .

posicion anterior(posicion p) const

Precondiciones: $L = (a_1, a_2, \dots, a_n, a_1)$

$1 \leq p \leq n$

Postcondiciones: Devuelve la posición anterior a p .

2. Una lista ordenada es una lista en la cual los elementos están ordenados de forma ascendente según la relación de orden definida entre ellos. Especifique e implemente el TAD *Lista ordenada*.
3. Implemente una función que tenga como entrada una lista de enteros y un entero x , de forma que devuelva la lista modificada mediante la eliminación de todas las ocurrencias de x en la lista.

4. Dadas dos listas de enteros ordenadas, implemente una función que devuelva una tercera lista ordenada con la mezcla de las dos listas de entrada.
5. Implemente una función *ImprimeInverso()* que imprima los elementos de una lista simplemente enlazada de enteros en orden inverso a partir de la posición p .
6. Se quiere representar el TAD *Conjunto* de tal forma que los elementos estén almacenados en una secuencia de celdas enlazadas.

Especifique e implemente el TAD *Conjunto*, representado mediante celdas enlazadas, con las siguientes operaciones: conjunto vacío, añadir un elemento al conjunto, quitar un elemento si pertenece al conjunto, unión, intersección y diferencia de conjuntos.

7. Implemente una función que concatene una lista de listas (TAD *Lista* cuyo tipo de elementos es a su vez una lista de elementos de cierto tipo T), de forma que a partir de la lista original, llamada *LInic* (lista de listas), se obtenga una nueva lista, llamada *LConcat* (lista de elementos T).
8. El TAD *número binario* consta de un conjunto de elementos definidos como secuencias de la forma $b_n b_{n-1} \dots b_1 b_0$, donde cada b_i es un dígito binario cuyo valor es 0 ó 1 y $n \geq 0$, que representan el valor numérico $\sum_{i=0}^n b_i 2^i$

Las operaciones de este TAD son las siguientes:

- Constructor: Genera un número binario a partir de una cadena de caracteres con ceros y unos.
- Operaciones lógicas de bits NOT, AND, OR inclusivo y OR exclusivo.
- Desplazamiento a la izquierda n bits. Cada bit se desplaza n posiciones a la izquierda, introduciendo ceros por la derecha y desechando los n bits de la izquierda.
- Desplazamiento a la derecha n bits. Se añaden n ceros por la izquierda y se desechan los n bits de la derecha.

- a) Realice la especificación del TAD *número binario* con las operaciones anteriores.
- b) Implemente el TAD *número binario* según la anterior especificación.

Nota 1: No conviene que estas operaciones sean miembros de la clase, sino amigas.

Nota 2: Las operaciones lógicas y los desplazamientos se pueden implementar mediante funciones, pero si lo prefiere puede sobrecargar los operadores \sim , $\&$, $|$, \wedge , $<<$ y $>>$.

9. En una variante de un juego bastante conocido, n jugadores, identificados por su nombre, se sitúan en círculo y se les asigna aleatoriamente un número del 1 al 6 (obviamente varios jugadores pueden tener asignado el mismo número). Se elige al azar un jugador de comienzo y, suponiendo que su número asignado es i , se elimina al jugador que está i posiciones hacia la izquierda si i es impar, o hacia la derecha si i es par. Se toma como nuevo valor de i el número que tuviera asignado el último jugador eliminado, el cuál, determina de la forma descrita el siguiente jugador a eliminar. Este proceso se repite hasta que quede un único jugador, el ganador del juego.

Implemente una función que determine y devuelva el jugador ganador, dado el conjunto de n jugadores con sus números del 1 al 6 asignados y la posición del jugador de comienzo.